Résumé Accéléré de JavaScript (JS)

1. 💡 Qu'est-ce que JavaScript ?

- **Le langage du Web :** JS est l'un des trois piliers du développement web, avec **HTML** (structure) et **CSS** (style).
- **Côté client (majoritairement) :** Il est exécuté par le **navigateur** pour rendre les pages web interactives (boutons, animations, formulaires dynamiques).
- **Côté serveur :** Grâce à **Node.js**, JS peut aussi être utilisé pour construire des serveurs et des applications backend.
- **Nature :** C'est un langage de script **interprété** (pas besoin de compilation) et **orienté objet** (même s'il est basé sur le concept de prototype).

. 📝 Syntaxe et Bases

- **Variables :** On utilise `let` et `const` (préférés à l'ancien `var`).
  - `const` : pour la déclaration de variable.
  - `let` : pour les valeurs qui peuvent être réassignées.
  - *Exemple :* `const PI = 3.14;` et `let compteur = 0;`
- **Types de Données souvent utilisés:**
  - **`string`** (texte, entre guillemets)
  - **`number`** (nombres entiers et décimaux)
  - **`boolean`** (`true` ou `false`)
- **Opérateurs de Comparaison Clés :**
  - == (égalité *lâche*, peut causer des problèmes de type, à éviter)
  - === (**égalité stricte**, compare la valeur **ET** le type) 👈 **À privilégier !**
  - `!=` (inégalité lâche) vs `!==` (**inégalité stricte**)

Opérateur	Nom	Que vérifie-t-il ?	L'erreur à éviter	Verdict
==	**Égalité Lâche**	**Valeur** (après conversion de type automatique, dite *coercition*).	10 == "10" donne **true**	**À ÉVITER** (Comportement imprévisible)
===	**Égalité Stricte**	**Valeur ET Type**. Aucune conversion n'est faite.	10 === "10" donne **false**	**À PRIVILÉGIER** (Clair et fiable)

"10" est pris comme une chaine de caractère

- **Structures de Contrôle :**
    - `if/else` : pour l'exécution conditionnelle.
    - `for`, `while` : pour les boucles (répétitions).

Exemple

----------**IF**----------

```
const TEMPERATURE = 28;
if (TEMPERATURE > 30) {
    console.log("Il fait très chaud !");
} else if (TEMPERATURE > 20) {
        console.log("La température est agréable.");
} else {
    console.log("Il fait frais.");
}
// SIMULATION CONSOLE (OUTPUT)
// La température est agréable.
```

----------**TERNAIRE**----------

```
const age = 19; const statut = (age >= 18) ? "Majeur" : "Mineur";
console.log("Vous êtes " + age)
// Ça affichera « Vous êtes Majeur»
```

----------**FOR**----------

```
// Affiche les nombres de 0 à 4
for (let i = 0; i < 5; i++) {
            console.log("Tour numéro : " + i);
}
 // SIMULATION CONSOLE (OUTPUT)
 Tour numéro : 0
Tour numéro : 1
Tour numéro : 2
Tour numéro : 3
Tour numéro : 4
```

----------**FOR**(moderne)----------

```
const fruits = ["pomme", "banane", "kiwi"];


// 'fruit' prend successivement la valeur de chaque élément du tableau
for (const fruit of fruits) {
    console.log("J'aime la " + fruit);
}


// SIMULATION CONSOLE (OUTPUT)
// J'aime la pomme
// J'aime la banane
// J'aime la kiwi
```

----------**WHILE**----------

⚠️ **Attention :** Il faut toujours s'assurer que quelque chose dans le bloc de code changera la condition pour qu'elle devienne fausse, sinon c'est une **boucle infinie** !

```
let tentativesRestantes = 3;


while (tentativesRestantes > 0) {
    console.log("Tentative(s) restante(s) : " + tentativesRestantes);
    tentativesRestantes--; // Décrémente pour que la boucle s'arrête
}


// SIMULATION CONSOLE (OUTPUT)
// Tentative(s) restante(s) : 3
// Tentative(s) restante(s) : 2
// Tentative(s) restante(s) : 1
```

3. ⚙️ Fonctions

Concept : Les fonctions sont des blocs de code réutilisables qui effectuent une tâche spécifique. Elles permettent d'éviter la répétition de code (**principe DRY**: *Don't Repeat Yourself*).

1. Déclaration Classique (Historique)

C'est la manière traditionnelle de définir une fonction.

```
// Fonction déclarée qui prend deux paramètres (a et b)
function soustraction(a, b) {
    return a - b; // 'return' renvoie le résultat au code appelant
}
 console.log(soustraction(10, 3));
// SIMULATION CONSOLE (OUTPUT)
// 7
```

Elle est toujours valide, mais la syntaxe fléchée est souvent préférée aujourd'hui.

2. Fonctions Fléchées (Arrow Functions) 🚀

C'est la syntaxe **moderne** et **compacte** (introduite avec ES6). Elle est très populaire car elle simplifie l'écriture, surtout pour les fonctions anonymes (callbacks).

A. Forme Complète

```
// On stocke la fonction dans une variable (const)
const multiplication = (a, b) => {
    // On peut mettre  plusieurs lignes d'instructions si l'utilité ce fait sentir
    return a * b;
};
console.log(multiplication(4, 5));
// SIMULATION CONSOLE (OUTPUT)
// 20
```

B. Forme Courte (Implied Return)

Si la fonction ne contient **qu'une seule instruction `return`**, on peut omettre les accolades `{}` et le mot-clé `return`. C'est le format le plus apprécié pour la concision !

```
const addition = (a, b) => a + b;
 console.log(addition(5, 5));
// SIMULATION CONSOLE (OUTPUT)
// 10
```

3. Les Fonctions Anonymes (Callbacks)

La fonction n'a pas de nom (pas de nom entre 'function' et '()')

```
const afficherMessage = function() {
    return "Je suis une fonction anonyme stockée dans 'afficherMessage'.";
};

// On l'appelle en utilisant le nom de la variable
console.log(afficherMessage());

// SIMULATION CONSOLE (OUTPUT)
// Je suis une fonction anonyme stockée dans 'afficherMessage'.
```

🧩 Objets et Tableaux (Collections de Données)

1. Les Tableaux (Arrays) : Représenter une Liste

Concept : Un Tableau est une liste **ordonnée** d'éléments. Chaque élément est accessible par son **index** (qui commence toujours à **0**).

Syntaxe : Utilisation de crochets `[]`.

```
const legumes = ["carotte", "brocoli", "poireau"];

// Accès par index (0 = premier élément)
console.log(legumes[0]); // Affiche "carotte"
console.log(legumes.length); // Affiche la taille du tableau

// Méthodes essentielles pour les Tableaux
legumes.push("courgette"); // Ajoute à la fin
legumes.pop(); // Retire le dernier élément

// SIMULATION CONSOLE (OUTPUT)
// carotte
// 3
```

2. Les Méthodes Clés des Tableaux (Haut Niveau) 🚀

Elles utilisent toutes des fonctions anonymes (callbacks) :

Méthode	Rôle	Exemple Rapide
.map()	**Transformation :** Crée un **nouveau tableau** en appliquant une fonction à chaque élément.	[1, 2].map(n => n * 2) $\ rightarrow$ [2, 4]
.filter()	**Filtrage :** Crée un **nouveau tableau** avec seulement les éléments qui remplissent une condition.	[5, 12, 8].filter(n => n > 10) $\rightarrow$ [12]
.forEach()	**Itération :** Exécute une fonction pour chaque élément (ne retourne rien).	noms.forEach(n => console.log(n))

3. Les Objets (Objects) : Représenter une Entité

Concept : Un Objet est une collection non ordonnée de paires **clé-valeur** (ou **propriété-valeur**). Il sert à modéliser des entités réelles (un utilisateur, une voiture, un produit, etc.).

Syntaxe : Utilisation d'accolades `{}`.

```
const utilisateur = {
    // Clé : Valeur
    prenom: "Marie",
    age: 28,
    ville: "Paris"
};
// Accès aux propriétés (la notation avec le point est la plus courante)
console.log(utilisateur.prenom);
console.log(utilisateur['age']);

// Modifier ou Ajouter une propriété
utilisateur.age = 29;
utilisateur.email = "marie@exemple.com";
// SIMULATION CONSOLE (OUTPUT)
// Marie
// 28
```